# A recovery mechanism for errors caused by a late subjob in a system handling SLA-based Grid workflows

Dang Minh Quan*

School of Information Technology
International University in Germany, Germany
E-mail: quandm@upb.de
*Corresponding author

Jörn Altmann

School of Information Technology
International University in Germany, Germany
and
Technology Management, Economics and Policy Program (TEMEP)
School of Engineering
Seoul National University, South Korea
E-mail: jorn.altmann@acm.org

**Abstract:** Supporting SLAs (Service Level Agreements) for Grid-based workflows requires providing mechanisms for handling errors (*i.e.*, the failures of subjobs). In the context of this paper, we propose an error recovery mechanism which can handle one failed subjob of a workflow. The error recovery mechanism has a maximum of three phases, depending on the impact of the error. In each phase, we use a dedicated algorithm to remap the subjobs of the workflow to the resources. The main contributions of the paper are the error recovery mechanism for SLA-based workflows and the mapping algorithm G-map, which is used in the first phase of the recovery mechanism. The G-map remaps the groups of subjobs, which are directly affected by an error. The efficiency of the proposed algorithm is validated through simulation results.

**Keywords:** Grid computing; Service Level Agreement; SLA; Grid-based workflow; error recovery.

**Reference** to this paper should be made as follows: Quan, D.M. and Altmann, J. (xxxx) 'A recovery mechanism for errors caused by a late subjob in a system handling SLA-based Grid workflows', *Int. J. Web and Grid Services*, Vol. X, No. Y, pp.000–000.

**Biographical notes:** Dang Minh Quan is a Senior Researcher at the School of Information Technology at the International University in Bruchsal. He received his PhD (2006) from the University of Paderborn, Germany. His current research centres on computer networks, high performance computing and Grid computing. In particular, he has put special focus on supporting the management of SLA-based workflows in the Grid.

Jörn Altmann is an Associate Professor for Techno-Economics at Seoul National University and an Associate Professor at the International University of Bruchsal, Germany, where he heads the group of Computer Networks and Distributed Systems. He received his PhD (1996) from the University of Erlangen-Nürnberg, Germany. His current research centres on the economics of internet services and internet infrastructures, integrating economic models into distributed systems. In particular, he focuses on capacity planning, network topologies and resource allocation.

## 1 Introduction

In the Grid computing environment, many users need the results of their calculations within a specific period of time. Examples of those users are a weather forecaster running weather forecasting workflows and an automobile producer running a dynamic fluid simulation workflow (Lovas *et al.*, 2004). Those users are willing to pay for getting their work completed on time. However, this requirement must be agreed on by both the users and the Grid provider before the application is executed. This agreement is kept in the Service Level Agreement (SLA) (Sahai *et al*., 2003). In general, SLAs are defined as an explicit statement of expectations and obligations in a business relationship between the service providers and the customers. SLAs specify the *a priori* negotiated resource requirements, the Quality of Service (QoS) and costs. The application of such an SLA represents a legally binding contract. This is a mandatory prerequisite for the next-generation Grids. We presented a prototype system supporting the SLAs for the Grid-based workflow in Quan and Kao (2005a–b), Quan and Hsu (2006) and Quan and Altmann (2007).

### *1.1 Grid-based workflow model*

Workflows received enormous attention in the databases and information systems research and development community (Elmagarmid, 1992; Georgakopoulos *et al*., 1995; Hsu, 1993). According to the definition from the Workflow Management Coalition (WfMC) (Fischer, 2004), a workflow is "The automation of a business process, in whole or parts, where documents, information or tasks are passed from one participant to another to be processed, according to a set of procedural rules." Although business workflows have great influence on research, another class of workflows emerged in sophisticated scientific problem-solving environments, which is called Grid-based workflows (Ludtke *et al*., 1999; Berriman *et al*., 2003; Lovas *et al*., 2004). A Grid-based workflow differs slightly from the WfMC definition, as it concentrates on intensive computation and data analysing, but not the business process. A Grid-based workflow is characterised by the following features (Singh and Vouk, 1997; Yu and Buyya, 2005):

- A Grid-based workflow usually includes many subjobs (*i.e.*, applications), which perform data analysis tasks. However, those subjobs are not executed freely, but in a strict sequence.

- A subjob in a Grid-based workflow depends tightly on the output data from the previous subjobs. With incorrect input data, a subjob will produce the wrong results and damage the result of the whole workflow.
- Subjobs in the Grid-based workflow are usually computation-intensive. They can be sequential or parallel programmes and require a long runtime.
- Grid-based workflows usually require powerful computing facilities (*e.g.*, supercomputers or clusters) to run on.

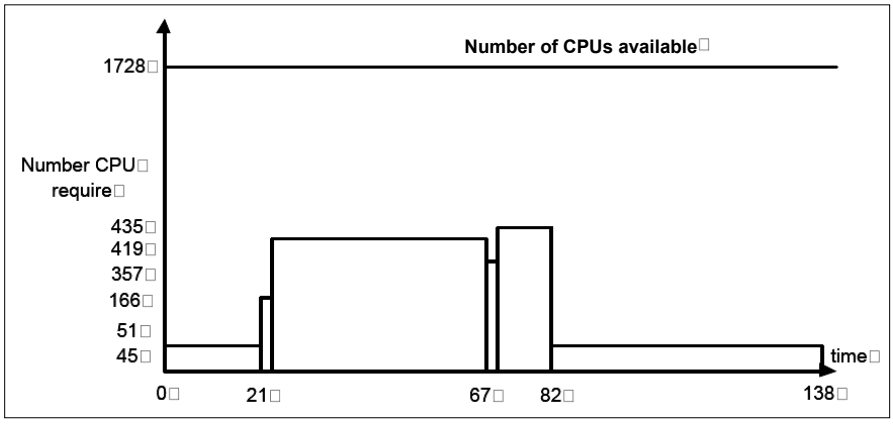
Like many popular systems handling Grid-based workflows (Deelman *et al*., 2004; Spooner *et al*., 2003; Lovas *et al*., 2004), our system is of the Directed Acyclic Graph (DAG) form. The user specifies the required resources needed to run each subjob, the data transfer between subjobs, the estimated runtime of each subjob and the expected runtime of the whole workflow.

In this paper, we assume that time is split into slots. Each slot equals a specific period of real time, from 3 to 5 min. We use the timeslot concept in order to limit the number of possible start times and end times of subjobs. Moreover, delaying for 3 min also has little meaning with the customer. It is noted that the data to be transferred between subjobs can be very large.

## *1.2 Grid service model*

The computational Grid includes many High Performance Computing Centres (HPCCs). The resources of each HPCC are managed by a software called the local Resource Management System (RMS).[1] Each RMS has its own unique resource configuration. A resource configuration comprises the number of Central Processing Units (CPUs), the amount of memory, the storage capacity, the software, the number of experts and the service price. To ensure that the subjob can be executed within a dedicated time period, the RMS must support advance resource reservation profile such as Computing Center Software (CCS) (Hovestadt, 2003). Figure 1 depicts an example of an CPU reservation profile of such an RMS. In our model, we reserve three main types of resources: CPU, storage and expert. The addition of further resources is straightforward.

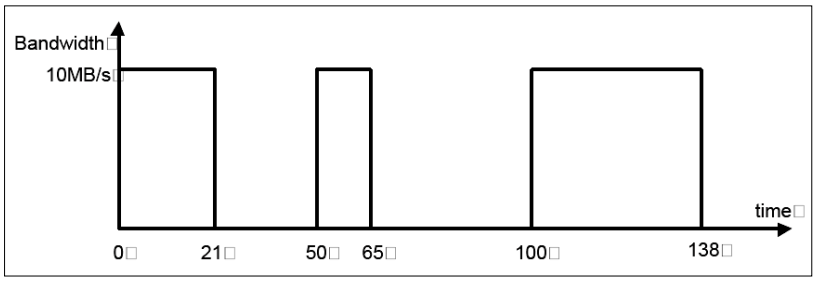
**Figure 1** A sample CPU reservation profile of a local RMS



If two output-input-dependent subjobs are executed on the same RMS, it is assumed that the time required for the data transfer equals zero. This can be assumed since all computed nodes in a cluster usually use a shared storage system like Network File System (NFS) or Distributed File System (DFS). In all other cases, it is assumed that a specific amount of data will be transferred within a specific period of time, requiring the reservation of bandwidth.

The link capacity between two local RMSs is determined as the average available capacity between those two sites in the network. The available capacity is assumed to be different for each different RMS couple. Whenever a data transfer task is required on a link, the possible time period on the link is determined. During that specific time period, the task can use the whole capacity and all the other tasks have to wait. Using this principle, the bandwidth reservation profile of a link will look similar to the one depicted in Figure 2. A more realistic model for bandwidth estimation (than the average capacity) can be found in Wolski (2003). Note that kind of bandwidth estimation model does not have any impact on the working of the overall mechanism.

**Figure 2** A sample bandwidth reservation profile of a link between two local RMSs




## *1.3 Business model*

In the case of a Grid-based workflow, letting users work directly with the resource providers has two main disadvantages:

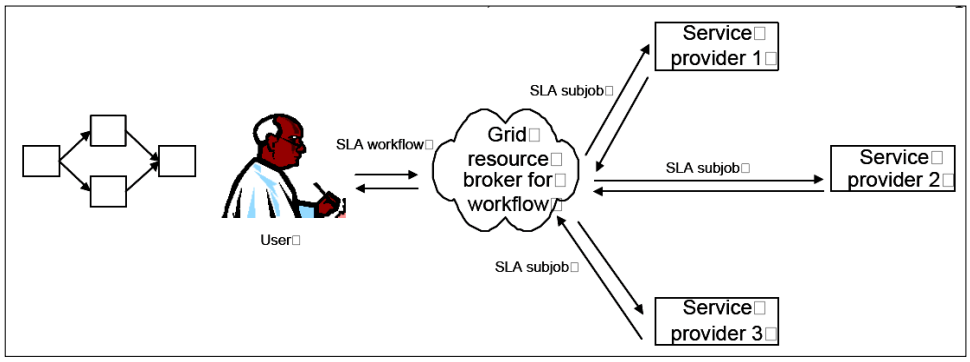
1 the user has to have sophisticated resource discovery and mapping tools in order to find the appropriate resource providers

2 the user has to manage the workflow, ranging from monitoring the running process to handling error events.

To free users from this kind of work, it is necessary to introduce a broker handling the workflow execution for the user (Altmann *et al*., 2007). We proposed a business model (Quan and Altmann, 2007) for the system, as depicted in Figure 3. There are three main entities: the end user, the SLA workflow broker and the service provider.

*The end user* wants to run a workflow within a specific period of time. The user asks the broker to execute the workflow for him and pays the broker for the workflow execution service. The user does not need to know in detail how much he has to pay each service provider; he only needs to know the total amount. This amount depends on the urgency of the workflow and the budget of the user. If there is an SLA violation, for example, the runtime deadline will not been met and the user will ask the broker for compensation. This compensation is clearly defined in the Service Level Objectives (SLOs) of the SLA.

**Figure 3** Stakeholders and their business relationship

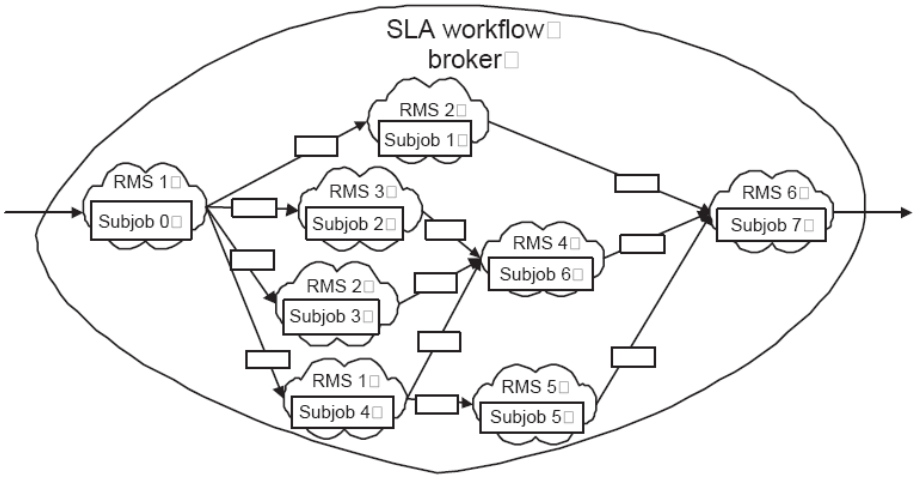



*The SLA workflow broker* represents the user, as specified in the SLA with the user. It controls the workflow execution. This includes the mapping of the subjobs to the resources, signing SLAs with the services providers, monitoring and error recovery. When the workflow execution is finished, it settles the accounts. It pays the service providers and charges the end user. The profit of the broker is the difference. The value-added that the broker provides is the handling of all the tasks for the end user.

*The service providers* execute the subjobs of the workflow. In our business model, we assume that each service provider fixes the price for its resources at the time of the SLA negotiation. As the resources of an HPCC usually have the same configuration and quality, each service provider has a fixed policy for compensation if its resources fail. For example, such a policy could be that $n\%$ of the cost will be compensated if the subjob is delayed for one time slot.

Figure 4 depicts a sample scenario of running a workflow on the Grid environment.

**Figure 4** A sample scenario running a Grid-based workflow



## 1.4 Problem statement

The error model that we assume is that an error inside an RMS may happen at any time during the subjob running period. The error could have been caused by an operating system failure, hardware failure or network cable failure. In all cases, the RMS will restart the subjob from the checkpoint image. We also assume that the time to detect the error and the time to rerun the subjob from the checkpoint image will cause the end time of the subjob to be later than the predetermined deadline. According to our business model, because of the fact that the provider is responsible for the failure, the late subjob will not be cancelled, but will be allowed to run a few additional timeslots. However, when one subjob is delayed, the output data transfer to the subsequent subjobs is delayed as well, causing the start time of those subjobs to be delayed. If those subjobs do not have the sufficient computational resources allocated to compensate for the shorter time available to complete their calculations, the original failure of the RMS might cause them to fail their calculations. Thus, it causes a cascading effect of failing subjobs. Therefore, the whole workflow will fail because of one single failure.

A concrete example of such a failure scenario is shown in Figure 4. If subjob 0 is delayed for one timeslot, the data transfer tasks 0–1, 0–2, 0–3 and 0–4 cannot be executed. Thus, subjobs 1, 2, 3 and 4 do not get the input data to start their calculation at the specified start times. The consequence is that the start of those subjobs will also be delayed by one timeslot. Those subjobs, however, might not have enough computational resources available to finish their calculation on time.

To avoid the delay of the whole workflow, the resource allocation of the subjobs of the workflow must be rescheduled so that they compensate the one timeslot delay. However, this rescheduling may bring some negative side effects.

- The finished time of the workflow may exceed the predetermined time period. The broker will be fined by the user according to the length of the delay.
- In rescheduling, if the remaining subjobs must be moved to the other RMSs, the broker has to cancel the old reservation contract. If this is not mentioned in the SLA, the broker will be fined by the service providers.

Therefore, it is desirable to have a mechanism to reschedule the subjobs of the workflow in such a way that the workflow can be executed to produce the final result while trying to keep the fines as low as possible. This paper presents an error recovery mechanism for SLA-based workflows that addresses these problems. The main contribution of the paper includes:

- An error recovery mechanism. The mechanism comprises three phases. The larger the impact of the error is, the more phases will be executed.
- An algorithm to remap a group of directly affected subjobs to the RMSs in a way that keeps the planned start time of the subjobs while minimising the costs.
- After an extensive experimental study, we found that if the delay is less than three timeslots, there is a large probability that the directly affected subjobs can be rescheduled successfully.

The paper is organised as follows. Section 2 describes the related work. Section 3 presents the error recovery mechanism. Section 4 describes the application of the existing mapping algorithm to the problem described above, while Section 5 introduces the proposed remapping algorithm, the G-map. The experiment about the quality of the proposed algorithms and the effect of the delay on the recovery is discussed in Section 6. Section 7 concludes the paper with a short summary.

## 2 Related works

Regarding the work on SLA support for Grid-based workflows, we have discussed the business models for brokers (Quan and Altmann, 2007), proposed mapping algorithms (Quan and Kao, 2005b; Quan and Hsu, 2006), defined an SLA language and built a prototype system (Quan and Kao, 2005a). Other works in this area focus on the QoS for Grid-based workflows (McGough *et al*., 2005; Zeng *et al*., 2004; Brandic *et al*., 2005). However, those works propose Grid resource service handling mechanisms, which can only handle one subjob at a time. These assumptions are not realistic, since many HPCCs provide their entire computing services under one single Grid resource service (Burchard *et al*., 2004).

Little work exists on the issue of error recovery for workflows, although the importance of fault tolerance in Grid computing has already been acknowledged with the establishment of the Grid Checkpoint Recovery Working Group. Its purpose is to define user-level mechanisms and Grid services for achieving fault tolerance. Stone (2004) described some initial results of the group's effort.

The well-known Condor system has also implemented a mechanism to handle errors (Condor Team, 2004). When the mechanism detects the error, it continues to execute the other subjobs of the workflow for as long as possible. This mechanism is reasonable if no SLAs have to be considered. Since it does not pay attention to meeting the deadline of a workflow, the cost incurred through fines and the need for extra resources can become very high.

Related to error recovery for Grid-based workflows within the SLA context, we described a mechanism coping with the catastrophic failure of one or several HPCCs in Quan (2007). The mechanism assumed that the failing subjob has to rerun. Thus, the probability to ensure the deadline of the workflow is very small. In Quan (2007), we proposed an algorithm which identifies those subjobs that are affected by the error and remaps those subjobs to the remaining healthy RMSs with makespan optimisation. In this paper, it has been assumed that the delay could be very small. This is the key factor of this paper. In particular, the differences are:

- In the case of fatal errors, the probability to ensure the deadline of a workflow is very small. Thus, the error recovery mechanism proposed in Quan (2007) only focuses on minimising the delay of the affected workflow. However, if a subjob is only delayed by a few timeslots, the probability to meet the deadline of the workflow is very high. Thus, we could focus on eliminating the extra cost. Trying to eliminate the delay is only the final attempt in this paper.

- The requirement of eliminating the extra cost leads to the problem of mapping a group of subjobs satisfying the deadline and optimising the cost. This issue does not appear in Quan (2007). This paper will propose an algorithm called the G-map to solve this problem.

The literature recorded a considerable amount of work in related areas, especially in finding the recovery methods for a single Grid job. Garbacki *et al.* (2005) presents a transparent fault tolerance for the Grid application based on Java Remote Method Invocation (RMI). They use a globally consistent checkpoint to avoid having to restart long-running computations from scratch after a system crash.

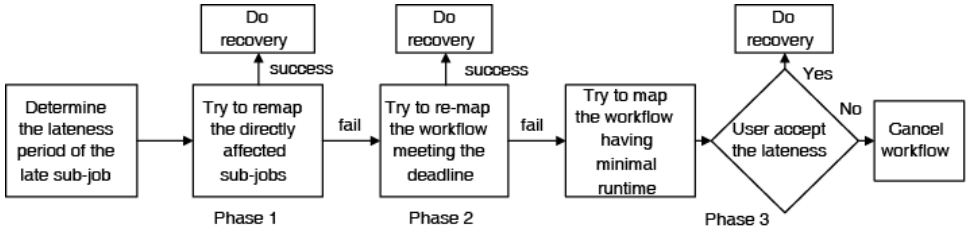
Hwang and Kesselman (2003) present a framework for handling failures on the Grid. Central to the framework is the flexibility in handling failures, which is achieved by using the workflow structure as a high-level recovery policy specification.

Heine *et al.* (2005a) describe an SLA-aware job migration mechanism in the Grid environments. The checkpoints of the running job can be migrated to the same or other clusters running Highly Predictable Cluster for Internet Grids (HPC4U) software (see Heine *et al*., 2005b). An architecture called Virtual Resource Management (VRM) manages the status of the process continuously.

## 3 The error recovery mechanism

In case of an error, we try to remap the remaining subjobs in a way that the workflow can be completed with little delay and little extra costs. The entire mechanism that includes three phases is described in Figure 5. Each phase represents a certain approach to find a remapping solution. The phases are sorted according to the simplicity and the cost that they incur.

**Figure 5** The error recovery mechanism for workflows in the case of one delayed subjob



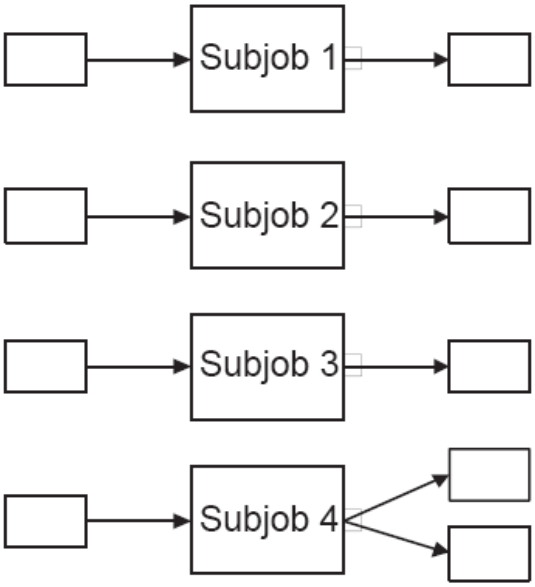

### *3.1 Phase 1: remapping the directly affected subjobs*

In the first phase, we will try to remap the directly affected subjobs in a way that does not affect the start time of the other remaining subjobs in the workflow. When we remap the directly affected subjobs, we also have to remap their related data transfers. For the example in Figure 4, if subjob 0 is delayed, the affected subjobs and data transfers are described in Figure 6. This task can be feasible for many reasons.

- The delay of the late subjob could be very small.
- The Grid may have others solution so that the data transfers will be shorter because the links have a broader bandwidth.

- The Grid may have RMSs with higher CPU power, which can execute the subjobs in a shorter time.

**Figure 6** Affected subjobs and data transfers




In the first phase, we try to adjust the execution time of the input data transfers, the affected subjobs and the output data transfers within the same RMS as predetermined. The subjobs which cannot be adjusted will be remapped to other RMSs. If this phase is successful, the broker only has to pay the following costs:

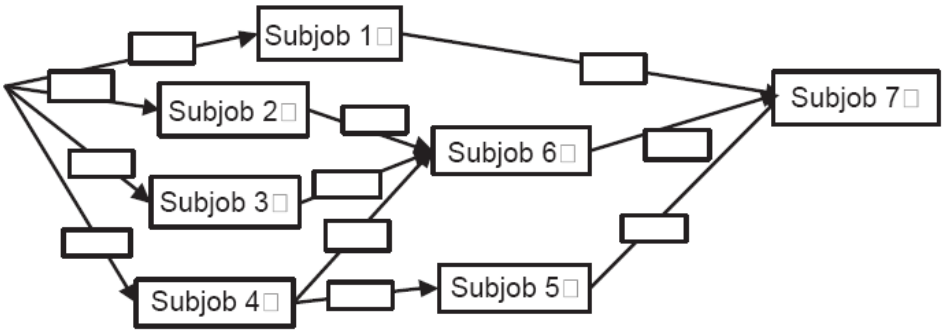
- the fee for cancelling the reserved resources of the directly affected subjobs
- the extra resource cost if the new mapping solution is more expensive than the old one.

As the cost for this phase is at least in the three phases, it should be tried first. The algorithms to remap the directly affected subjobs are described in more detail in Sections 4 and 5.

### *3.2 Phase 2: remapping the workflow to meet the predetermined deadline*

This phase is executed if the first phase was not successful. In this phase, we will try to remap the remaining workflow in a way that the deadline of the workflow is met and the cost is minimised. The remaining workflow is formed from all the non-executed subjobs by using the conventional graph algorithm. A more detailed description about this task can be found in Quan (2007). In the case of our example, if subjob 0 is delayed, the remaining workflow comprises all subjobs and their data transfers are as illustrated in Figure 7.

**Figure 7** The workflow needs to be remapped




If this phase is successful, the broker has to pay the following costs:

- the fee for cancelling the reserved resources of all the remaining subjobs
- the extra resource cost if the new mapping solution is more expensive than the old one.

As this problem is similar to the problem of mapping a Grid-based workflow within the SLA context, we use the H-map algorithm to find the solution. The detailed description about the H-map algorithm can be seen in Quan and Hsu (2006).

### *3.3 Phase 3: remapping the workflow to have minimal runtime*

This phase is the final attempt to recover from the error. It is initiated if the two previous phases were not successful. In this phase, we try to remap the remaining workflow in a way that minimises the delay of the entire workflow. If the solution has an acceptable lateness, the broker has to pay the following costs:

- the fee for cancelling the reserved resources of all the remaining subjobs
- the extra resource cost if the new mapping solution has a higher cost than the old one
- the fine for finishing the entire workflow late. This cost increases proportionally with the length of the delay.

If the algorithm only finds a solution with a delay higher than what is accepted by the user, the whole workflow will be cancelled and the broker has to pay the following costs:

- the fee for cancelling the reserved resources of all the remaining subjobs
- the fine for not finishing the entire workflow.

The goal of this phase equals to minimising the total runtime of the workflow. To do the remapping, we use the w-Tabu algorithm, which was described in Quan (2007).

## 4 Mapping the directly affected subjobs by applying the existing mapping algorithms

Mapping the directly affected subjobs can be described by the following specifications.

- Let $R$ be the set of Grid RMSs. This set includes a finite number of RMSs, which provide static information about the controlled resources and the current reservations/assignments.
- Let $S$ be the set of subjobs in the given workflow.
- Let $Sa$ be the set of all directly affected subjobs with the resource and runtime requirements.
- Let $E$ be the set of data transfer in the given workflow.
- Let $Ei$ be the set of input data transfers of $S$.
- Let $Eo$ be the set of output data transfers of $S$.
- Let $K_i$ be the set of resource candidates of subjob $s_i$. This set includes all the RMSs, which can run subjob $s_i$, $K_i \subset R$, $s_i \in Sa$.

Based on the given input, a feasible and possibly optimal solution is sought by allowing the most efficient mapping of those subjobs in a Grid environment with respect to the given deadlines. The required solution is a set defined as Formula 1.

$$M = \{s_i, r_j, start\_slot) \mid s_i \in Sa, r_j \in K_i\}. \tag{1}$$

If the solution does not have *start_slot* for each $s_i$, it become a configuration, as defined in Formula 2.

$$a = \{(s_i, r_j \mid s_i \in Sa, r_j \in K_i\}. \tag{2}$$

A feasible solution must satisfy the following conditions:

Criteria 1 All $K_i \neq \emptyset$. There is at least one RMS in the candidate set of each subjob.

Criteria 2 The start time of each input data transfer $ei_h$ must be later than the subjob it depends on, $ei_h \in Ei$. The stop time of each output data transfer $eo_k$ must be earlier than the next subjob which depends on it, $eo_k \in Eo$.

Criteria 3 Each RMS provides a profile of currently available resources and can run many subjobs, both sequentially and in parallel. Those subjobs, which run on the same RMS, form a profile of the resource requirement. With each RMS $r_j$ running the subjobs of $Sa$ with each timeslot in the profile of available resources and the profile of resource requirements, the number of the available resources must be larger than the resource requirements.

Criteria 4 The data transmission task $e_{ki}$ from subjob $s_k$ to subjob $s_i$ must not overlap the other reserved data transmission tasks on the link between the RMS running subjob $s_k$ to the RMS running subjob $s_i$, $e_{ki} \in Ei \cup Eo, s_k, s_i \in Sa$.

In the next phase, the most feasible solution with the lowest cost is sought. The cost C of a Grid workflow is defined in Formulae 3, 4 and 5. It is a sum of four factors: the cost of (1) using the CPU, (2) storage, (3) expert knowledge and (4) the data transfer between resources.

$$C_1 = \sum_{i=1}^{n} s_i \cdot r_t * (s_i \cdot n_c * r_j \cdot p_c + s_i \cdot n_s * r_j \cdot p_s + s_i \cdot n_e * r_j \cdot p_e) \tag{3}$$

$$C_2 = \sum e_{ki} \cdot n_d * r_j \cdot p_d \tag{4}$$

$$C = C_1 + C_2, \tag{5}$$

with $s_i \cdot r_t$, $s_i \cdot n_c$, $s_i \cdot n_s$, $s_i \cdot n_e$ being the runtime, number CPU, number storage and number expert of subjob $s_i$, respectively. $r_j \cdot p_c$, $r_j \cdot p_s$, $r_j \cdot p_e$, $r_j \cdot p_d$ are the prices of using the CPU, storage, expert knowledge and data transmission of RMS $r_j$, respectively. $e_{ki} \cdot n_d$ is the number of data to be transferred from subjob $s_k$ to subjob $s_i$.

If two output-input-dependent subjobs are executed under the same RMS, the cost of transferring data from the previous subjob to the later subjob is neglected. It can be easily shown that the optimal mapping of those subjobs to the Grid RMSs with cost optimising is an NP-hard problem.

Our problem can be defined as a special case of the mapping of a bag of independent tasks to the resources problem (Sulistio and Buyya, 2005; Gao *et al.*, 2005; Braun *et al.*, 2001). However, there are significant differences between our problem and the formal one.

- In our problem, the subjobs are parallel applications and the resources are RMSs, while in the formal problems, the tasks in the sequential programmes and the resources are computing nodes.
- While the goal of our problem is optimising the cost, the goal of the formal problem is optimising the execution time.
- While the input and output data transfers in our problem play an important role, they are usually not considered in the formal problem.

For those reasons, in the first step, we have to adapt some existing algorithms to our problems. In this part, we will present the application of the Search All Cases (SAC), the H-map and the Deadline and Budget-Constrained (DBC) algorithms.

### *4.1 The SAC algorithm*

The SAC algorithm is presented in Figure 8. The algorithm will consider all the possible configurations to check for the validation and the cost to find the best one. Within the SAC algorithms, the module determining the scheduling order and the module checking the feasibility are described in Parts 5.3 and 5.4. The main problem with this algorithm is the slow speed. If the number of affected subjobs is more than five and the number of RMSs is more than ten, the runtime of the algorithm may take several hours. This is not acceptable in practice. However, we will still apply the SAC algorithm because the number of affected subjobs is not always big. In the case of having a small number of affected subjobs, the runtime of the algorithm is tolerable. The result of this algorithm can be a good reference source to evaluate the quality of the other algorithms.

**Figure 8** The SAC algorithm

```
min_cost=1000000
With each possible configuration a {
   determine the scheduling order
   check the feasibility of a
   if feasible {
      compute the cost m_cost of a
      if(m_cost<min_cost){
            min_cost=m_cost
            store a
} } }
```

### *4.2 The H-map algorithm*

In Quan and Hsu (2006), we present an algorithm called the H-map to map heavy communication workflows to the Grid resources. The main idea of the H-map algorithm is that a set of initial configurations distributed over the search space according to the cost factor will be further refined to find the best solution. To form the set of initial configurations, the candidate RMSs of each subjob are sorted in cost order. A configuration is formed by selecting an RMS at a ranking level. Each configuration is then improved by using a local search. The character of the resources and the workload are similar to this problem. Therefore, we can easily adapt the H-map to the problem, as described in Figure 9. In the H-map algorithm, the procedure to determine the timetable and the order of the schedule time are described in Parts 5.3 and 5.4.

**Figure 9** The application of the H-map algorithm to the problem

```
Create set of reference configurations distributed over the
search space according to cost factor
for each sub-job in the set {
   for each RMS in the candidate list {
      if cheaper then put (sjid, RMS id, improve_value)
      to a list }}
sort the list according to improve_value
from the begin of the list{
   Compute time table to get the finished time
   If finished time < limit
      break
}
Store the result
```

### 4.3 The DBC algorithm

The closest work to our problem is from Buyya *et al.* (2005). In Buyya *et al.* (2005), the authors present the method to schedule parameter sweep applications on global Grids. The original DBC cost time optimisation algorithm builds on the cost optimisation and time optimisation scheduling algorithms. This is accomplished by applying the time optimisation algorithm to schedule the task-farming application jobs on the distributed resources having the same processing cost. For each subjob and for each RMS in the sorted candidate list, if the RMS satisfies the deadline, then we select that RMS and continue with the other subjobs. The authors assume that all tasks are sequential programmes and identical to each other. It is clear that our problem is distinguished from parameter sweep applications scheduling problems, as the subjobs in our problem are parallel programmes with various differences in configurations as well as in constraints. However, the primary idea of the DBC algorithm can also be applied to our problem, as presented in Figure 10.

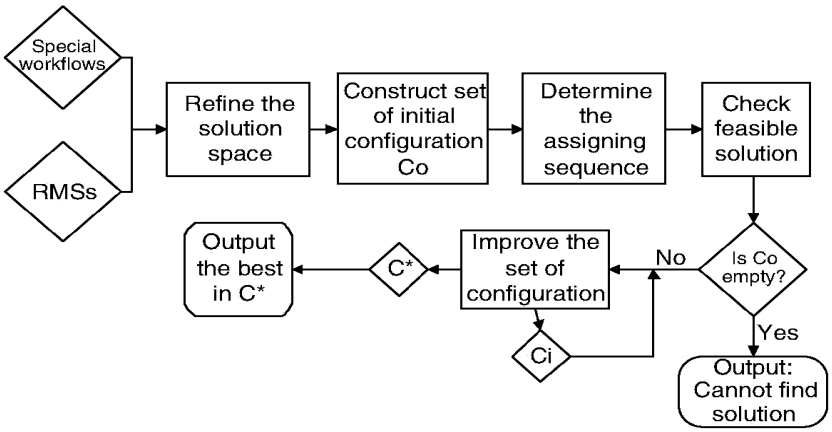
**Figure 10** The application of the DBC algorithm to our problem

```
For each sub-job in the set {
  Sort the candidate RMSs according to cost order. If 2 or
   more RMSs have the same cost, the more powerful RMS
   will stay first
   For each RMS in the sorted candidate list {
      calculate the execution time of that sub-job on the
       RMS. If it meet the deadline then assigned the sub-
       job to the RMS
}}
```

## 5 The G-map algorithm

The G-map algorithm maps a group of subjobs onto the Grid resources, where G stands for Group. In the G-map algorithm, we try to compress the solution space in a way that the ability for feasible solutions is higher. After that, a set of initial configurations is constructed. This set will be improved by a local search until it cannot be improved any more. Finally, we pick the best solution from the final set. The architecture of the algorithm is presented in Figure 11.

**Figure 11** The G-map algorithm




## *5.1 Refining the solution space*

The set of candidate RMSs for each subjob can be continuously refined by the following observation: an RMS will be valid with a subjob only if the subjob assigned to that RMS satisfies the start time of the next sequential subjobs. The algorithm to refine the solution space is presented in Figure 12.

**Figure 12** Refining the solution space procedure

```
for each sub-job k in the set {
  for each RMS r in the candidate list of k{
    for each link to k in assigned sequence{
      min_st_tran=end_time of source sub-job
      search reservation profile of link  the
      start_tran > min_st_tran
      end_tran = start_tran+num_data/bandwidth
      update reservation profile
    }
    min_st_sj=max (end_tran)
    search in reservation profile of  r the
    start_job > min_st_sj
    end_job= start_job + runtime
    for each link from k in assigned sequence{
      min_st_tran=end_job
      search reservation profile of link  the
      start_tran > min_st_tran
      end_tran = start_tran+num_data/bandwidth
      update reservation profile
      if end_tran>=end_time of destination sub-job
        remove r       out of the candidate list
}}}
```

With each separate subjob, we determine the schedule time of the input data transfers, the subjob and the output data transfer. From the algorithm in Figure 12, we can see that the resource reservation profile is not updated. We call this the ideal assignment. If the stop time of the output data transfer is not earlier than the start time of the next sequential subjob, then we remove the RMS from the candidate set.

### *5.2 Constructing the set of initial configurations*

The goal of the algorithm is to find a feasible solution which satisfies all the required criteria and is as inexpensive as possible. Therefore, the set of initial configurations should satisfy two criteria.

1 The configurations in the set must differ from each other as far as possible. This criterion will ensure that the set of initial configurations will be widely distributed over the search space.

2 The RMS running the subjob in each configuration should differ from each other. This criterion will ensure that each subjob will be assigned in the ideal condition; thus, the ability to become a feasible solution will be increased.

The procedure to create the set of initial configurations is as follows.

Step 1 *Sorting the candidate set according to the cost factor.* With each subjob, we compute the cost of running the subjob by each RMS in the candidate set and then sort the RMSs according to the cost. In the case of our example, we could have a sorted solution space, as presented in Figure 13.

**Figure 13** A sample sorted solution space

| | | | | |
|---|---|---|---|---|
| Subjob 1 | 2 | 4 | 1 | 3 |
| Subjob 2 | 1 | 2 | 3 | 4 |
| Subjob 3 | 4 | 2 | 1 | 3 |
| Subjob 4 | 1 | 3 | 4 | 2 |

Step 2 *Forming the first configuration.* The procedure to form the first configuration in the set is presented in Figure 14. We form the first solution with as small a cost as possible. With each unassigned subjob, we compute the *m_delta* = the cost running in the first feasible RMS minus the cost running in the second feasible RMS in the sorted candidate list. The subjob having the smallest *m_delta* will be assigned to the first feasible RMS. The purpose of this action is to ensure that the subjob having the higher ability of increasing the cost will be assigned first. After that, we will update the reservation profile and check if the assigned RMS is still available for other subjobs. If not, we will mark it as unavailable. This process is repeated until all the subjobs are assigned. The selection of which subjob to assign is effective when there are as many subjobs having the same RMS as the first feasible solution.

**Figure 14** The algorithm to form the first configuration

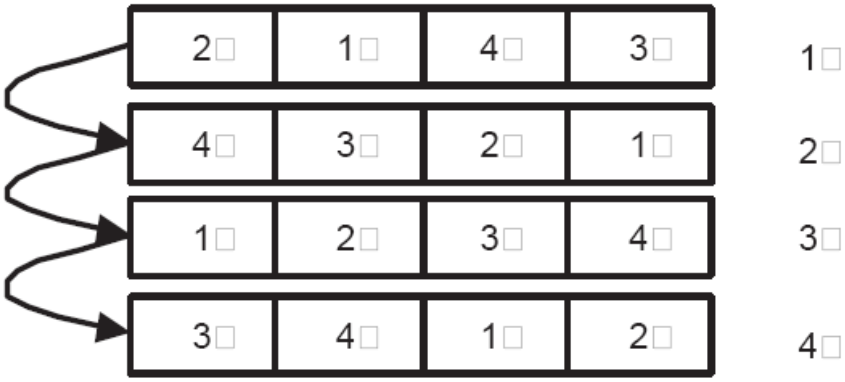
```
While the set of unassigned sub-jobs is not empty {
   Foreach sub-job s in the set of unassigned sub-jobs {
      m_delta=cost in first feasible RMS- cost in second
      feasible RMS
      put (s, RMS, m_delta) in a list
   }
   Sort the list to get the minimum m_delta
   Assign s to the RMS
   Drop s out of the set of unassigned sub-jobs
   Update the reservation profile of the RMS
   Check if the RMS is still feasible with other unassigned
   sub-jobs
   if not, mark the RMS is infeasible
}
```

Step 3 *Forming the other configurations.* The procedure to form the other initial configurations is described in Figure 15. To satisfy the two criteria as described above, we use *assign_number* to keep track of the number of the assignment of RMS to a subjob and *l_ass* to keep track of the appearance frequency of the RMS within a configuration. The RMS having the smaller *assign_number* and the smaller appearance frequency in *l_ass* will be selected. Applying this algorithm to the example, starting from the first configuration in Step 2, the process of forming other initial configurations is described in Figure 16.

**Figure 15** The procedure to create the initial configuration set

```
assign_number of each candidate RMS =0
While number of configuration < max_sol {
  clear list of assigned RMS l_ass
  for each sub-job in the set {
     find in the candidate list RMS r having the
     smallest number of appearance in l_ass
     and the smallest assign_number
     Put r to l_ass
     assign_number++
}}
```

**Figure 16** The sample initial configuration set




### *5.3 Determining the assigning order*

When the RMS executing each subjob and the bandwidths among the subjobs have been determined, the next task is determining the timeslot to run a subjob in the specific RMS. At this time, the order of determining the scheduled time for subjobs becomes important. The sequence of determining the runtime for the subjobs in the RMS can also affect Criteria 2, especially in the case of having many subjobs in the same RMS. In this algorithm, we use the policy like in Quan and Hsu (2006). Thus, the input data transfer having the smaller earliest start time will be scheduled earlier. The output data transfer having the smaller latest stop time will be scheduled earlier. The subjob having the earlier deadline should be scheduled earlier.

### *5.4 Checking the feasibility of a solution*

To check the feasibility of a solution, we have to determine the timetable to execute the subjobs and their related data transfer. In the error recovery phase, finding a solution that meets Criteria 2 is very important. Therefore, we do not simply use the provided runtime of each subjob, but modify it according to the performance of each RMS. Let $pk_i$, $pk_j$ be the performance of a CPU in RMS $r_i$, $r_j$, respectively, and $pk_j > pk_i$. Suppose that a subjob has the provided runtime $rt_i$ with the resource requirement equal to $r_i$. Thus, the runtime $rt_j$ of the subjob in $r_j$ is determined in Formula 6.

$$rt_j = \frac{rt_i}{\frac{pk_i + (pk_j - pk_i) * k}{pk_i}}. \tag{6}$$

Parameter $k$ presents the effect of the subjob's communication character and the RMS's communication infrastructure. For example, if $pk_j$ equals to $2*pk_i$ and the $rt_i$ is 10 h, $rt_j$ will be 5 h if $k$ equals one. However, $k = 1$ only when there is no communication among the parallel tasks of the subjob. Otherwise, $k$ will be less than 1.

The practical Grid workflow usually has a fixed input data pattern. For example, the weather forecasting workflow is executed day by day and finishes within a constant period of time since all the data was collected (Lovas *et al.*, 2004). This character is the

basis for estimating the Grid workload's runtime (Spooner *et al.*, 2003). In our paper, parameter $k_a$ is an average value, which is determined by the user through many experiments and is provided as the input for the algorithm.

In the real environment, $k$ may fluctuate around the average value, depending on the network infrastructure of the system. For example, suppose that $k_a$ equals 0.8. If the cluster has good network communications, the real value of $k$ may increase to 0.9. If the cluster has not-so-good network communications, the real value of $k$ may decrease to 0.7. Nowadays, with very good network technology in the HPCCs, the fluctuation of $k$ is not so much. To overcome the fluctuation problem, we use the pessimistic value $k_p$ instead of $k$ in Formula 6 to determine the new runtime of the subjob, as follows.

- If $k_a > 0.8$, for example, with the rare communication subjob, $k_p = 0.5$.
- If $0.8 > k_a > 0.5$, for example, with the normal communication subjob, $k_p = 0.25$.
- If $k_a < 0.5$, for example, with a heavy communication subjob, $k_p = 0$.

The pessimistic policy will ensure that the subjob can be finished within the new determined runtime period. With those assumptions, the procedure to determine the timetable is presented in Figure 17.

**Figure 17** The procedure to determine the timetable

```
for each sub-job k in the set {
  for each link to k in assigned sequence{
     min_st_tran=end_time of source sub-job
     search reservation profile of link  the
     start_tran > min_st_tran
     end_tran = start_tran+num_data/bandwidth
     update link reservation profile
  }
  min_st_sj=max (end_tran)
  search in reservation profile of RMS running
  k the start_job > min_st_sj
  end_job= start_job + runtime
  update resource reservation profile
  for each link from k in assigned sequence{
    min_st_tran=end_job
    search reservation profile of link  the
    start_tran > min_st_tran
    end_tran = start_tran+num_data/bandwidth
    update link reservation profile
  }}
```

After determining the timetable, the stop time of the output data transfer will be compared with the start time of the next sequential subjobs. If there is a violation, this solution is determined as infeasible.

### *5.5 Improving the solution quality algorithm*

If the initial configuration set is $C_0 \neq \varnothing$, the set will gradually be refined to have better quality solutions. The refining process stops when the solutions in the set cannot be improved any more and we have the final set C*. The best solution in *C** will be the output as the result of the algorithm. More details about this procedure can be found in Quan and Hsu (2006).

## 6 Performance experiment

The performance experiment is done with a simulation to check for the quality of the G-map algorithm and determine the effect of the late period to the recovery process. We use simulation data because we want to cover a wide range of the character of the workload, which is impossible with a real workload. The hardware and software used in the experiments is rather standard and simple (Pentium D, 2.8 GHz, 1 GB RAM, Fedora Core 5, MySQL). The total simulation programme includes about 5000 lines of C/C++ code. To do the experiment, we generated eight different workflows, which:

- Have different topologies.
- Have a different maximum number of potentially directly affected subjobs. The number of subjobs is in the range of one to ten. The number of the potentially directly affected subjobs stops at ten because as far as we know with the workload model as described in Part 1, this number in the real workflow is just between one and seven.
- Have different subjob specifications. Without the loss of generality, we assume that each subjob has the same CPU performance requirement.
- Have a different amount of data transfer.

As the difference in the static factors of an RMS, such as the Operating System (OS), the CPU speed and so on, can be easily filtered by an SQL query, we use 20 RMSs with the resource configuration equal to or even better than the requirement of subjobs. Those RMSs have already had some initial workload in their resource reservation profiles and bandwidth reservation profiles. Those eight workflows are mapped to 20 RMSs. We select the late subjob in each workflow in a way that the number of the directly affected subjobs equals the maximum number of the potentially directly affected subjobs of that workflow. The late period is one time slot. With each group of the affected subjobs, we change the power configuration of the RMS and the *k* value of affected subjobs. Those configurations are presented in Table 6. For example, with the first row in Table 6, the resource configuration 90-0-10 means that 90% of the RMSs have a CPU performance like the requirement, 0% RMSs having a CPU performance 25% more than the requirement and 10% of RMS have a CPU performance 50% more than the requirement. The workload configuration 90-0-10 means that 90% of the affected subjobs have

$k = 0.5$, 0% of the affected subjobs have $k = 0.25$ and 10% of the affected subjobs have $k = 0$. From Table 1, we can see that the RMS configuration spreads in a wide range from having many RMSs with a more powerful CPU to having many RMSs having a CPU equal to the requirement. We can also see that the workload configuration changes widely from having many subjobs with a big $k$ to having many subjobs with a small $k$. We have chosen this experiment schema because we want to study the character of the algorithm in all possible cases.

**Table 1** Resource configuration scenario and workload configuration scenario

| *Resource* | | | | *Configuration* | |
|---|---|---|---|---|---|
| *0%* | *Configuration 25%* | *50%* | *Workload k = 0.5* | *k = 0.25* | *k = 0* |
| 90 | 0 | 10 | 90 | 0 | 10 |
| 90 | 10 | 0 | 90 | 10 | 0 |
| 60 | 30 | 10 | 60 | 30 | 10 |
| 60 | 10 | 30 | 60 | 10 | 30 |
| 30 | 60 | 10 | 30 | 60 | 10 |
| 30 | 10 | 60 | 30 | 10 | 60 |
| 10 | 30 | 60 | 10 | 30 | 60 |
| 10 | 60 | 30 | 10 | 60 | 30 |
| 10 | 0 | 90 | 10 | 0 | 90 |
| 10 | 90 | 0 | 10 | 90 | 0 |
| 0 | 10 | 90 | 0 | 10 | 90 |
| 0 | 90 | 10 | 0 | 90 | 10 |

## *6.1 The quality of algorithms*

The goal of the experiment is to compare the quality of the G-map algorithm with other algorithms under different workload and resource configurations. The quality of an algorithm is evaluated by several factors: the ability of finding a feasible solution, the cost of the solution found and the execution time. To evaluate the quality of the G-map algorithm, we set the late period to one timeslot. With each affected subjob group, with each power resource configuration scenario and with each workload configuration scenario, we do the mapping with four algorithms: the G-map, DBC, H-map and SAC. Thus, with each algorithm, we have a total of 8*12*12 = 1152 instances. For each instance, we record the runtime of the algorithm and the cost of the solution, if it is feasible.

Table 2, 3 and 4 present the detailed cost and runtime of each algorithm for the different groups of affected subjobs in three extreme experimental scenarios. Table 2 describes the data of the scenario with a more powerful RMS than the requirement, a resource configuration of 90-10-0 and a lot of subjobs having a big $k$ value and a workload configuration at 0-10-90. Table 3 describes the data of the scenario with the average number of the more powerful RMSs than the requirement, a resource configuration of 60-30-0 and an average number of subjobs having a big $k$ value and a workload configuration of 30-60-0. Table 4 describes the data of the scenario with a lot of RMSs equal to the requirement, a resource configuration of 0-10-90, a lot of subjobs having a small $k$ value and a workload configuration of 90-10-0.

**Table 2** The experimental result with a resource configuration of 90-10-0 and a workload configuration of 0-10-90

| *Sjs* | *SAC cost* | *Rt* | *H-map cost* | *Rt* | *DBC cost* | *Rt* | *G-map cost* | *Rt* |
|---|---|---|---|---|---|---|---|---|
| 3 | 50.3 | 4 | 50.3 | 0.5 | 50.3 | 0.5 | 50.3 | 0.5 |
| 4 | 62.0 | 59 | 62.0 | 1 | 62.0 | 1 | 62.0 | 1 |
| 5 | 87.7 | 1211 | 87.7 | 1 | 87.7 | 1 | 87.7 | 1 |
| 6 | – | – | 78.4 | 1 | 81.5 | 1 | 78.5 | 1 |
| 7 | – | – | 95.1 | 1 | 102.3 | 1 | 95.1 | 1 |
| 8 | – | – | 113.5 | 1 | 113.5 | 1 | 113.5 | 1 |
| 9 | – | – | 110.7 | 1 | 119.7 | 1 | 110.7 | 1 |
| 10 | – | – | 144.6 | 1 | 150.6 | 1 | 142.8 | 1 |

**Table 3** The experimental result with a resource configuration of 60-30-0 and a workload configuration of 30-60-0

| *Sjs* | *SAC cost* | *Rt* | *H-map cost* | *Rt* | *DBC cost* | *Rt* | *G-map cost* | *Rt* |
|---|---|---|---|---|---|---|---|---|
| 3 | 50.3 | 4 | 50.3 | 0.5 | 50.3 | 0.5 | 50.3 | 0.5 |
| 4 | N/S | 60 | N/S | 1 | N/S | 1 | N/S | 1 |
| 5 | 87.7 | 1217 | 87.7 | 0.5 | 87.7 | 1 | 87.7 | 1 |
| 6 | – | – | 81.5 | 1 | 92 | 0.5 | 81.5 | 1 |
| 7 | – | – | 95.1 | 1 | 102.3 | 1 | 95.1 | 0.5 |
| 8 | – | – | N/S | 0.5 | 113.5 | 0.5 | 113.5 | 1 |
| 9 | – | – | N/S | 1 | N/S | 1 | N/S | 0.5 |
| 10 | – | – | N/S | 0.5 | 166.2 | 1 | 153.6 | 1 |

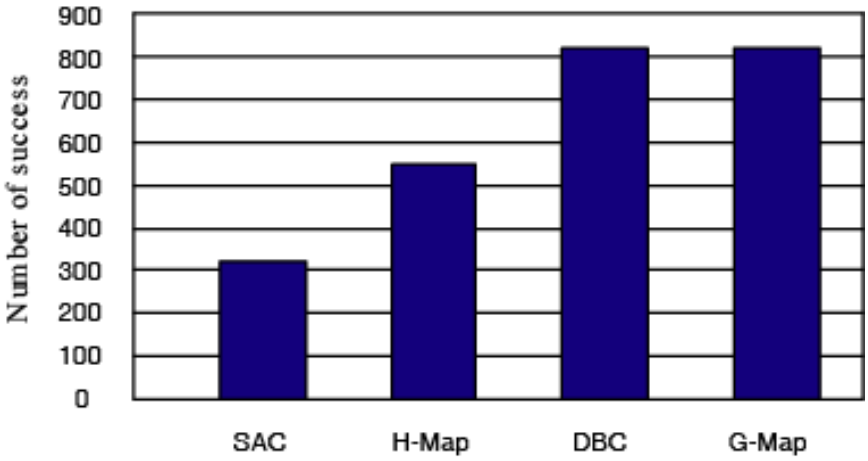
**Table 4** The experimental result with a resource configuration of 0-10-90 and a workload configuration of 90-10-0

| *Sjs* | *SAC cost* | *Rt* | *H-map cost* | *Rt* | *DBC cost* | *Rt* | *G-map cost* | *Rt* |
|---|---|---|---|---|---|---|---|---|
| 3 | 50.3 | 4 | 50.3 | 0.5 | 50.3 | 0.5 | 50.3 | 0.5 |
| 4 | N/S | 57 | N/S | 1 | N/S | 1 | N/S | 1 |
| 5 | 87.7 | 1205 | 87.7 | 0.5 | 87.7 | 1 | 87.7 | 1 |
| 6 | – | – | 81.5 | 1 | 92 | 0.5 | 81.5 | 1 |
| 7 | – | – | 95.1 | 1 | 102.3 | 1 | 95.1 | 1 |
| 8 | – | – | N/S | 0.5 | N/S | 0.5 | N/S | 1 |
| 9 | – | – | N/S | 1 | N/S | 1 | N/S | 1 |
| 10 | – | – | N/S | 0.5 | 166.2 | 1 | 153.6 | 1 |

Among the four algorithms, only the SAC algorithm has a very long runtime when the size of the problem increases. The runtime of this algorithm increases exponentially if the number of the affected subjobs is greater than or equal to six. Other algorithms have very short runtimes. In all cases, the runtime of the H-map, DBC and G-map is not greater than one second.

From the data in the tables, we can see clearly a trend that if the Grid has a lot of powerful RMSs than required and the group of affected subjobs has a lot of computing-intensive jobs (big $k$), the chance of having a feasible solution will be greater and vice versa. The SAC algorithm can find a solution within an acceptable period when the size of the problem is small. Therefore, it has the highest chance of finding a feasible solution. The H-map algorithm has a limited chance of finding a feasible solution. The reason is that the H-map is designed for mapping the whole workflow and the step of refining the solution space is not performed. Thus, there are a lot of infeasible solutions in the initial configuration set. The G-map and the DBC algorithm have the same ability for finding a feasible solution. Figure 18 presents the total number of finding out the feasible solutions in the whole experiment corelative with each algorithm.

**Figure 18** The total number of feasible found solutions by the algorithms




From the experiment data, we can see the domination of the local search approach with respect to the quality of the solution. If the H-map or G-map algorithms can find a feasible solution, it is a high quality solution at a low cost. The SAC and the H-map algorithms have a smaller chance of finding a feasible solution. We compare only the quality of the solution between the G-map and the DBC algorithms. Figure 19 presents the average cost in relative value between the two algorithms. We can see that the G-map finds a lower cost solution than the DBC.

## *6.2 The effect of the late period to the recovery process*

To evaluate the effect of the late period on the recovery process, we change the lateness period from one timeslot to five timeslots. For each affected subjob group, for each power resource configuration scenario, for each workload configuration scenario and for each late period, we perform the whole recovery process with the G-map, the H-map and

the w-Tabu algorithm. If the G-map algorithm in Phase 1 is not successful, the H-map algorithm in Phase 2 is revoked. If the H-Map is not successful, the w-Tabu algorithm in Phase 3 is revoked. Thus, for each late period, we have a total of 8*12*12=1152 recovery instances. For each late period, we record the number of feasible solutions for each algorithm and also for each phase of the recovery process. The experimental data is presented in Figure 20.

**Figure 19** The average cost comparison in relative value

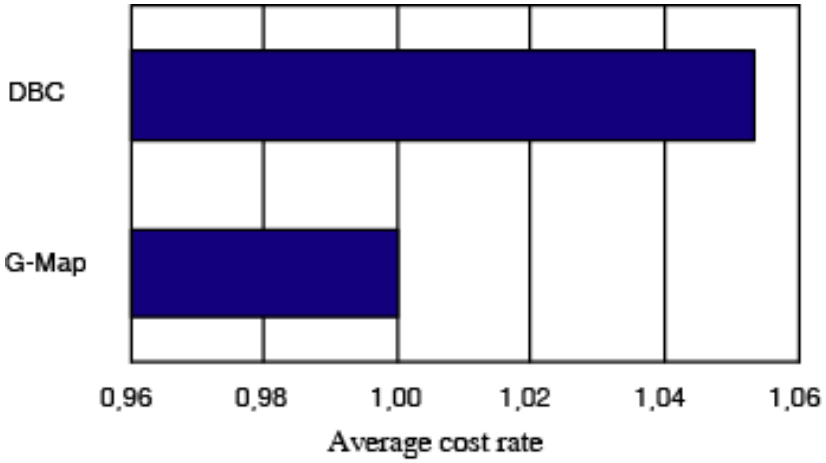
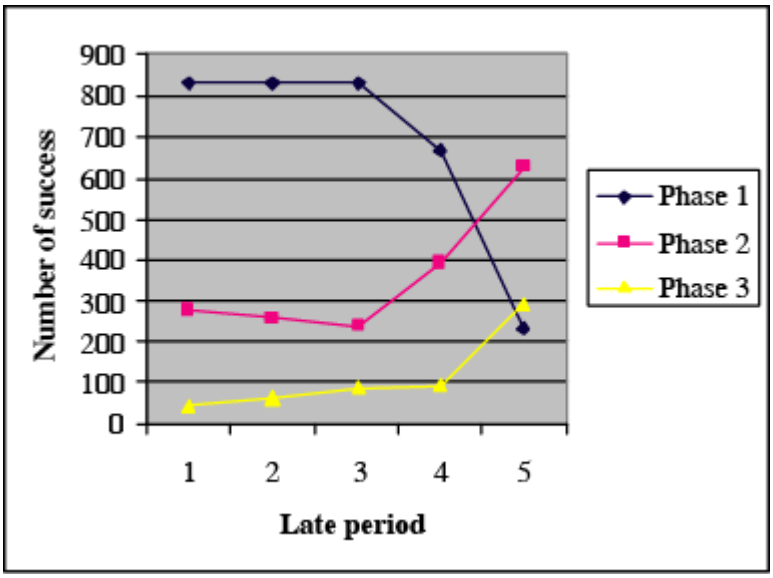



**Figure 20** The number of successful recoveries in each phase




From Figure 20, we can see that the error is effectively recovered when the late period is between one and three timeslots. If the late period is less than or equal to three timeslots, the ability to successfully recover with a low cost by the first phase is very high at 830

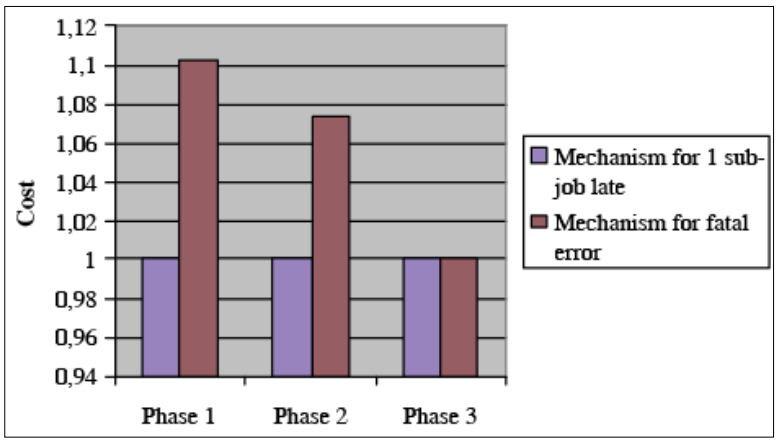
times out of 1152. When the late period is greater than three, the chance of failure of Phase 1 increases sharply and we have to invoke the second phase or third phase, whichever has the higher cost.

### *6.3 The performance of the error recovery mechanism*

In this section, we compare the performance of the error recovery mechanism presented in this paper with the work in Quan (2007). The performance of an error recovery mechanism is defined as the cost that the broker has to pay for the negative effect of the error, as described in Part 3. If the cost is smaller, the performance of the mechanism is better and vice versa.

To do the experiment, we set the lateness period to one. Each reserved resource cancellation costs 10% of the resource hiring value. For each affected subjob group, for each power resource configuration scenario and for each workload configuration scenario, we execute both recovery mechanisms and record the cost and the phase in which the error recovery mechanism for one late subjob is successful. For each phase, we compute the average relative cost of the successful solutions found by both mechanisms. The final result is depicted in Figure 21.

**Figure 21** The performance comparison between two error recovery mechanisms




The proposed error recovery in this paper includes three phases. Phase 3 is the same as the work in Quan (2007b). Therefore, if Phase 3 is invoked, the performance of the mechanism in this paper is equal to the performance of the mechanism in Quan (2007). This fact is confirmed in Figure 21.

If Phase 2 or Phase 1 is successful, the performances of the two mechanisms are different. From Figure 21, we can see that if the error recovery mechanisms for one late subjob succeeds at Phase 1 or 2, the broker will pay less money than when using the mechanism in Quan (2007). Figure 20 shows that this probability is large when the delay is small. Therefore, in any case, the performance of the mechanism described in this paper is at least equal or better than the one in Quan (2007).

## 7 Conclusion

This paper has described an error recovery mechanism for the errors caused by one delayed subjob of a SLA-based Grid workflow. The first contribution of the paper is the newly stated problem and the error recovery mechanism to address the problem. The error recovery mechanism comprises three phases to cope with different constraints. In the first phase, the method tries to remap the directly affected subjobs in a way that does not affect the start time of the other remaining subjobs in the workflow. If the first phase is not successful, the method tries to remap the remaining workflow in a way that the deadline of the workflow is met while keeping the cost as low as possible. If the second step is not successful as well, the method tries to remap the remaining workflow in a way that the lateness of the workflow is minimised. The second contribution is the cost-minimising mapping algorithm, named G-map, to remap a group of the affected subjobs to the RMSs. We also adapted many recently appeared algorithms to this problem and compared it with the G-map algorithm in order to prove the improved performance of the G-map algorithm. The experimental results also show that the proposed algorithm finds cost-effective solutions within a short runtime. As the quality of the error recovery depends mainly on the remapping algorithms, these two characteristics ensure a fast and effective error recovery procedure.

## Acknowledgements

The authors would like to acknowledge the many helpful suggestions of two anonymous reviewers and the participants of the AINA 2007 Conference on the earlier versions of this paper. We also thank Professor Fatos Xhafa, Professor Runhe Huang and Professor Hai Jin, the Editors. The work described in this paper has been supported through the GridEcon project by the European Commission.

## References

Altmann, J., Ion, M. and Mohammed, A.A.B. (2007) 'Taxonomy of Grid business models', *Proceedings of the 4th International Workshop on Grid Economics and Business Models*, pp.29–43.

Berriman, G.B., Good, J.C. and Laity, A.C. (2003) 'Montage: a Grid enabled image mosaic service for the National Virtual Observatory', *AD ASS*, Vol. 13, pp.145–167.

Brandic, I., Benkner, S., Engelbrecht, G. and Schmidt, R. (2005) 'QoS support for time-critical Grid workflow applications', *Proceedings of the First International Conference on e-Science and Grid Computing,* pp.108–115.

Braun, T.D., Siegel, H.J., Beck, N., Blni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., *et al*. (2001) 'A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems', *Journal of Parallel and Distributed Computing,* Vol. 61, No. 6, pp.810–837.

Burchard, L., Hovestadt, M., Kao, O., Keller, A. and Linnert, B. (2004) 'The virtual resource manager: an architecture for SLA-aware resource management', *Proceedings of the IEEE CCGrid 2004*, IEEE Press, pp.126–133.

Buyya, R., Murshed, M., Abramson, D. and Venugopal, S. (2005) 'Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimisation algorithm', *Software: Practice and Experience (SPE)*, Vol. 35, No. 5, pp.491–512.

Condor Team (2004) 'CondorVersion 6.4.7 manual', www.cs.wisc.edu/condor/manual/v6.4 (10 December 2004).

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K. and Livny, M. (2004) 'Pegasus: mapping scientific workflows onto the Grid', *Proceedings of the 2nd European Across Grids Conference*, pp.11–20.

Elmagarmid, A.K. (1992) *Database Transaction Models for Advanced Applications*, Morgan Kaufmann.

Fischer, L. (2004) *Workflow Handbook 2004*, Future Strategies Inc., Lighthouse Point, Florida, USA.

Gao, Y., Rong, H. and Huang, J.Z. (2005) 'Adaptive Grid job scheduling with genetic algorithms', *Future Gener. Comput. Syst.*, Vol. 21, No. 1, pp.151–161.

Garbacki, P., Biskupski, B. and Bal, H. (2005) 'Transparent fault tolerance for Grid application', *Proceedings of the European Grid Conference (EGC 2005)*, LNCS 3470, pp.671–680.

Georgakopoulos, D., Hornick, M. and Sheth, A. (1995) 'An overview of workflow management: from process modeling to workflow automation infrastructure', *Distributed and Parallel Databases*, Vol. 3, No. 2, pp.119–153.

Heine, F., Hovestadt, M., Kao, O. and Keller, A. (2005a) 'Provision of fault tolerance with Grid-enabled and SLA-aware resource management systems', *Proceeding of the Parallel Computing Conference 2005*, pp.105–112.

Heine, F., Hovestadt, M., Kao, O. and Keller, A. (2005b) 'SLA-aware job migration in Grid environments', *Grid Computing: New Frontiers of High Performance Computing*, Elsevier Press, pp.345–367.

Hovestadt, M. (2003) 'Scheduling in HPC resource management systems:queuing vs. planning', *Proceedings of the 9th Workshop on JSSPP at GGF8*, LNCS, pp.1–20.

Hsu, M. (Ed.) (1993) *Special Issue on Workflow and Extended Transaction Systems*, IEEE Data Engineering, Vol. 16, No. 2.

Hwang, S. and Kesselman, C. (2003) 'GridWorkflow: a flexible failure handling framework for the Grid', *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, IEEE Press, pp.126–131.

Lovas, R., Dzsa, G., Kacsuk, P., Podhorszki, N. and Drtos, D. (2004) 'Workflow support for complex Grid applications: integrated and portal solutions', *Proceedings of 2nd European Across Grids Conference*, pp.129–138.

Ludtke, S., Baldwin, P. and Chiu, W. (1999) 'EMAN: semiautomated software for high-resolution single-particle reconstruction', *Journal of Structure Biology*, Vol. 128, pp.146–157.

McGough, S., Afzal, A., Darlington, J., Furmento, N., Mayer, A. and Young, L. (2005) 'Making the Grid predictable through reservations and performance modelling', *The Computer Journal*, Vol. 48, No. 3, pp.358–368.

Quan, D.M. (2007) 'Error recovery mechanism for Grid-based workflow within SLA context', *Int. J. High Performance Computing and Networking*, Vol. 5, Nos. 1–2, pp.110–121.

Quan, D.M. and Altmann, J. (2007) 'Business model and the policy of mapping light communication Grid-based workflow within the SLA context', *Proceedings of the International Conference of High Performance Computing and Communication (HPCC07)*, pp.285–295.

Quan, D.M. and Hsu, D.F. (2006) 'Network based resource allocation within SLA context', *Proceedings of the GCC2006*, pp.274–280.

Quan, D.M. and Kao, O. (2005a) 'Mapping Grid job flows to Grid resources within SLA context', *Proceedings of the European Grid Conference (EGC 2005)*, LNCS 3470, pp.1107–1116.

Quan, D.M. and Kao, O. (2005b) 'On architecture for an SLA-aware job flows in Grid environments', *Journal of Interconnection Networks*, Vol. 6, No. 3, pp.245–264.

Sahai, A., Graupner, S., Machiraju, V. and Moorsel, A. (2003) 'Specifying and monitoring guarantees in commercial Grids through SLA', *Proceeding of the 3rd IEEE/ACM CCGrid2003*, pp.292–300.

Singh, M.P. and Vouk, M.A. (1997) 'Scientific workflows: scientific computing meets transactional workflows', http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html.

Spooner, D.P., Jarvis, S.A., Cao, J., Saini, S. and Nudd, G.R. (2003) 'Local Grid scheduling techniques using performance prediction', *IEEE Proceedings – Computers and Digital Techniques*, pp.87–96.

Stone, N. (2004) 'GWD-I: An architecture for Grid checkpoint recovery services and a GridCPR API', http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-Architecture-2.0.pdf (10 December 2004).

Sulistio, A. and Buyya, R. (2005) 'A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems', *Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp.235–242.

Wolski, R. (2003) 'Experiences with predicting resource performance on-line in computational Grid settings', *ACM SIGMETRICS Performance Evaluation Review*, Vol. 30, No. 4, pp.41–49.

Yu, J. and Buyya, R. (2005) 'A taxonomy of scientific workflow systems for Grid computing', *SIGMOD Record*, Vol. 34, No. 3, pp.44–49.

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J. and Chang, H. (2004) 'QoS-aware middleware for web services composition', *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, pp.311–327.

## Note

1 In this paper, the RMS is used to represent the cluster/supercomputer as well as the Grid service provided by the HPCC.